ABOUT FAHD SHARIFF

Fahd is a software engineer working in the financial services industry. He is passionate about technology and specializes in Java application development in distributed environments.

Analysing a Java Core Dump

Posted by: Fahd Shariff in Core Java February 21st, 2013

In this post, I will show you how you can debug a Java core file to see what caused your JVM to crash. I will be using a core file I generated in my previous post: Generating a Java Core Dump. There are different ways you can diagnose a JVM crash, listed below:

The hs_err_pid log file

When a fatal error occurs in the JVM, it produces an error log file called

```
hs_err_pidXXXX.log
```

, normally in the working directory of the process or in the temporary directory for the operating system. The top of this file contains the cause of the crash and the 'problematic frame'. For example, mine shows:

```
$ head hs_err_pid21178.log
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x0000002b1d00075c, pid=21178, tid=1076017504
#
# JRE version: 6.0_21-b06
# Java VM: Java HotSpot(TM) 64-Bit Server VM (17.0-b16 mixed mode linux-amd64 )
# Problematic frame:
# C  [libnativelib.so+0x75c]  bar+0x10
#
```

There is also a stack trace:

```
Stack: [0x000000004012b000,0x000000004022c000],  sp=0x000000004022aac0,  free space=3fe0000000000000018k
Native frames: (J=compiled Java code, j=interpreted, Vv=VM code, C=native code)
C  [libnativelib.so+0x75c]  bar+0x10
C  [libnativelib.so+0x772]  foo+0xe
C  [libnativelib.so+0x78e]  Java_CoreDumper_core+0x1a
j  CoreDumper.core()V+0
j  CoreDumper.main([Ljava/lang/String;)V+7
v  ~StubRoutines::call_stub
V  [libjvm.so+0x3e756d]
```

The stack trace shows that my java method,

```
CoreDumper.core()
```

, called into JNI and died when the

```
bar
```

function was called in native code.

Debugging a Java Core Dump

In some cases, the JVM may not produce a

```
hs_err_pid
```

file, for example, if the native code abruptly aborts by calling the

```
abort
```

function. In such cases, we need to analyse the core file produced. On my machine, the operating system writes out core files to

```
/var/tmp/cores
```

. You can use the following command to see where your system is configured to write out core files to:

```
$ cat /proc/sys/kernel/core_pattern
/var/tmp/cores/%e.%p.%u.core
$ ls /var/tmp/cores
java.21178.146385.core
```

There are a few, different ways to look at core dumps:

GNU Debugger (gdb) can examine a core file and work out what the program was doing when it crashed.

```
$ gdb $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
(gdb) where
#0  0x0000002a959bd26d in raise () from /lib64/tls/libc.so.6
#1  0x0000002a959bea6e in abort () from /lib64/tls/libc.so.6
#2  0x0000002b1cecf799 in bar () from libnativelib.so
#3  0x0000002b1cecf7a7 in foo () from libnativelib.so
#4  0x0000002b1cecf7c3 in Java_CoreDumper_core () from libnativelib.so
#5  0x0000002a971aac88 in ?? ()
#6  0x0000000040113800 in ?? ()
#7  0x0000002a9719fa42 in ?? ()
#8  0x000000004022ab10 in ?? ()
#9  0x0000002a9a4d5488 in ?? ()
#10 0x000000004022ab70 in ?? ()
#11 0x0000002a9a4d59c8 in ?? ()
#12 0x0000000000000000 in ?? ()
```

The

```
where
```

command prints the stack frames and shows that the

```
bar
```

function called

```
abort()
```

which caused the crash.

2. Using jstack

```
jstack
```

prints stack traces of Java threads for a given core file.

```
$ jstack -J-d64 $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
Debugger attached successfully.
Server compiler detected.
JVM version is 17.0-b16
Deadlock Detection:

No deadlocks found.

Thread 16788: (state = BLOCKED)

Thread 16787: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.ref.ReferenceQueue.remove(long) @bci=44, line=118 (Interpreted frame)
 - java.lang.ref.ReferenceQueue.remove() @bci=2, line=134 (Interpreted frame)
 - java.lang.ref.Finalizer$FinalizerThread.run() @bci=3, line=159 (Interpreted frame)

Thread 16786: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.Object.wait() @bci=2, line=485 (Interpreted frame)
 - java.lang.ref.Reference$ReferenceHandler.run() @bci=46, line=116 (Interpreted frame)

Thread 16780: (state = IN_NATIVE)
 - CoreDumper.core() @bci=0 (Interpreted frame)
 - CoreDumper.main(java.lang.String[]) @bci=7, line=12 (Interpreted frame)
```

3. Using jmap

```
jmap
```

examines a core file and prints out shared object memory maps or heap memory details.

```
$ jmap -J-d64 $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
Debugger attached successfully.
Server compiler detected.
JVM version is 17.0-b16
0x0000000040000000      49K     /usr/sunjdk/1.6.0_21/bin/java
```

```
0x0000002a9566c000      124K    /lib64/tls/libpthread.so.0
0x0000002a95782000      47K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/jli/libjli.so
0x0000002a9588c000      16K     /lib64/libdl.so.2
0x0000002a9598f000      1593K   /lib64/tls/libc.so.6
0x0000002a95556000      110K    /lib64/ld-linux-x86-64.so.2
0x0000002a95bca000      11443K  /usr/sunjdk/1.6.0_21/jre/lib/amd64/server/libjvm.so
0x0000002a96699000      625K    /lib64/tls/libm.so.6
0x0000002a9681f000      56K     /lib64/tls/librt.so.1
0x0000002a96939000      65K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/libverify.so
0x0000002a96a48000      228K    /usr/sunjdk/1.6.0_21/jre/lib/amd64/libjava.so
0x0000002a96b9e000      109K    /lib64/libnsl.so.1
0x0000002a96cb6000      54K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/native_threads/libhpi.so
0x0000002a96de8000      57K     /lib64/libnss_files.so.2
0x0000002a96ef4000      551K    /lib64/libnss_db.so.2
0x0000002a97086000      89K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/libzip.so
0x0000002b1cecf000      6K      /home/sharfah/tmp/jni/libnativelib.so
```

Useful Links:

Crash course on JVM crash analysis

Reference: Analysing a Java Core Dump from our JCG partner Fahd Shariff at the fahd.blog blog.

Tagged with: JVM

Do you want to know how to develop your skillset to become a Java Rockstar?

Subscribe to our newsletter to start Rocking right now!

To get you started we give you our best selling eBooks for FREE!

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more ....

Email address:

Your email address

Sign up

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

✓ Sign me up for the newsletter!

Receive Email Notifications?

no, do not subscribe ▾ instantly ▾

Or, you can subscribe without commenting.

Post Comment

KNOWLEDGE BASE

- Courses
- Tutorials
- Whitepapers

PARTNERS

- Mkyong

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

HALL OF FAME

- “Android Full Application Tutorial” series
- Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons
- Android Google Maps Tutorial
- Android JSON Parsing with Gson Tutorial
- Android Location Based Services Application – GPS location
- Android Quick Preferences Tutorial
- Difference between Comparator and Comparable in Java
- GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
- Java Best Practices – Vector vs ArrayList vs HashSet

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused o
ultimate Java to Java developers resource center; targeted at the techn

source projects.

GET THE JAVA SKILLS YOU NEED IN 2016. Start Learn

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geel
property of their respective owners. Java is a trademark or registered t
Oracle Corporation in the United States and other countries. Examples J
is not connected to Oracle Corporation and is not sponsored by Oracle (